错误处理（Error Handling）

2.1 翻译+校对：lyojo (https://github.com/lyojo) ray16897188 (https://github.com/ray16897188) 2015-10-23 校对：shanks (http://codebuild.me) 2015-10-24

2.2 翻译+校对：SketchK (https://github.com/SketchK) 2016-05-15

本页包含内容：

*错误处理（Error handling）*是响应错误以及从错误中恢复的过程。Swift 提供了在运行时对可恢复错误的抛出、捕获、传递和操作的一流支持。

某些操作无法保证总是执行完所有代码或总是生成有用的结果。可选类型可用来表示值缺失，但是当某个操作失败时，最好能得知失败的原因，从而可以作出相应的应对。

举个例子，假如有个从磁盘上的某个文件读取数据并进行处理的任务，该任务会有多种可能失败的情况，包括指定路径下文件并不存在，文件不具有可读权限，或者文件编码格式不兼容。区分这些不同的失败情况可以让程序解决并处理某些错误，然后把它解决不了的错误报告给用户。

> 注意
> Swift 中的错误处理涉及到错误处理模式，这会用到 Cocoa 和 Objective-C 中的 `NSError` 。关于这个类的更多信息请参见 Using Swift with Cocoa and Objective-C (Swift 2.2) (https://developer.apple.com/library/prerelease/ios/documentation/Swift/Conceptual/BuildingCocoaApps/index.htm 中的错误处理 (https://developer.apple.com/library/prerelease/ios/documentation/Swift/Conceptual/BuildingCocoaApps/Adopting CH7-ID10)。

表示并抛出错误

在 Swift 中，错误用符合 `ErrorType` 协议的类型的值来表示。这个空协议表明该类型可以用于错误处理。

Swift 的枚举类型尤为适合构建一组相关的错误状态，枚举的关联值还可以提供错误状态的额外信息。例如，你可以这样表示在一个游戏中操作自动贩卖机时可能会出现的错误状态：

```
enum VendingMachineError: ErrorType {
    case InvalidSelection                    //选择无效
    case InsufficientFunds(coinsNeeded: Int) //金额不足
    case OutOfStock                          //缺货
}
```

抛出一个错误可以让你表明有意外情况发生，导致正常的执行流程无法继续执行。抛出错误使用 `throw` 关键字。例如，下面的代码抛出一个错误，提示贩卖机还需要 5 个硬币：

```
throw VendingMachineError.InsufficientFunds(coinsNeeded: 5)
```

处理错误

某个错误被抛出时，附近的某部分代码必须负责处理这个错误，例如纠正这个问题、尝试另外一种方式、或是向用户报告错误。

Swift 中有 4 种处理错误的方式。你可以把函数抛出的错误传递给调用此函数的代码、用 `do-catch` 语句处理错误、将错误作为可选类型处理、或者断言此错误根本不会发生。每种方式在下面的小节中都有描述。

当一个函数抛出一个错误时，你的程序流程会发生改变，所以重要的是你能迅速识别代码中会抛出错误的地方。为了标识出这些地方，在调用一个能抛出错误的函数、方法或者构造器之前，加上 `try` 关键字，或者 `try?` 或 `try!` 这种变体。这些关键字在下面的小节中有具体讲解。

> 注意
>
> Swift 中的错误处理和其他语言中用 `try`， `catch` 和 `throw` 进行异常处理很像。和其他语言中（包括 Objective-C ）的异常处理不同的是，Swift 中的错误处理并不涉及解除调用栈，这是一个计算代价高昂的过程。就此而言， `throw` 语句的性能特性是可以和 `return` 语句相媲美的。

用 throwing 函数传递错误

为了表示一个函数、方法或构造器可以抛出错误，在函数声明的参数列表之后加上 `throws` 关键字。一个标有 `throws` 关键字的函数被称作*throwing 函数*。如果这个函数指明了返回值类型， `throws` 关键词需要写在箭头（ `->` ）的前面。

```
func canThrowErrors() throws -> String
func cannotThrowErrors() -> String
```

一个 throwing 函数可以在其内部抛出错误，并将错误传递到函数被调用时的作用域。

> 注意
>
> 只有 throwing 函数可以传递错误。任何在某个非 throwing 函数内部抛出的错误只能在函数内部处理。

下面的例子中， `VendingMechine` 类有一个 `vend(itemNamed:)` 方法，如果请求的物品不存在、缺货或者投入金额小于物品价格，该方法就会抛出一个相应的 `VendingMachineError`：

```
struct Item {
    var price: Int
    var count: Int
}

class VendingMachine {
    var inventory = [
        "Candy Bar": Item(price: 12, count: 7),
        "Chips": Item(price: 10, count: 4),
        "Pretzels": Item(price: 7, count: 11)
    ]
    var coinsDeposited = 0
    func dispenseSnack(snack: String) {
        print("Dispensing \(snack)")
    }

    func vend(itemNamed name: String) throws {
        guard let item = inventory[name] else {
            throw VendingMachineError.InvalidSelection
        }

        guard item.count > 0 else {
            throw VendingMachineError.OutOfStock
        }

        guard item.price <= coinsDeposited else {
            throw VendingMachineError.InsufficientFunds(coinsNeeded: item.price - coinsDeposit
ed)
        }

        coinsDeposited -= item.price

        var newItem = item
        newItem.count -= 1
        inventory[name] = newItem

        dispenseSnack(name)
    }
}
```

在 `vend(itemNamed:)` 方法的实现中使用了 `guard` 语句来提前退出方法，确保在购买某个物品所需的条件中，有任一条件不满足时，能提前退出方法并抛出相应的错误。由于 `throw` 语句会立即退出方法，所以物品只有在所有条件都满足时才会被售出。

因为 `vend(itemNamed:)` 方法会传递出它抛出的任何错误，在你的代码中调用此方法的地方，必须要么直接处理这些错误——使用 `do-catch` 语句， `try?` 或 `try!`；要么继续将这些错误传递下去。例如下面例子中， `buyFavoriteSnack(_:vendingMachine:)` 同样是一个 throwing 函数，任何由 `vend(itemNamed:)` 方法抛出的错误会一直被传递到 `buyFavoriteSnack(_:vendingMachine:)` 函数被调用的地方。

```
let favoriteSnacks = [
    "Alice": "Chips",
    "Bob": "Licorice",
    "Eve": "Pretzels",
]
func buyFavoriteSnack(person: String, vendingMachine: VendingMachine) throws {
    let snackName = favoriteSnacks[person] ?? "Candy Bar"
    try vendingMachine.vend(itemNamed: snackName)
}
```

上例中， `buyFavoriteSnack(_:vendingMachine:)` 函数会查找某人最喜欢的零食，并通过调用 `vend(itemNamed:)` 方法来尝试为他们购买。因为 `vend(itemNamed:)` 方法能抛出错误，所以在调用的它时候在它前面加了 `try` 关键字。

throwing构造器能像throwing函数一样传递错误.例如下面代码中的 `PurchasedSnack` 构造器在构造过程中调用了throwing函数,并且通过传递到它的调用者来处理这些错误。

```
struct PurchasedSnack {
    let name: String
    init(name: String, vendingMachine: VendingMachine) throws {
        try vendingMachine.vend(itemNamed: name)
        self.name = name
    }
}
```

用 Do-Catch 处理错误

可以使用一个 `do-catch` 语句运行一段闭包代码来处理错误。如果在 `do` 子句中的代码抛出了一个错误，这个错误会与 `catch` 子句做匹配，从而决定哪条子句能处理它。

下面是 `do-catch` 语句的一般形式：

```
do {
    try expression
    statements
} catch pattern 1 {
    statements
} catch pattern 2 where condition {
    statements
}
```

在 `catch` 后面写一个匹配模式来表明这个子句能处理什么样的错误。如果一条 `catch` 子句没有指定匹配模式，那么这条子句可以匹配任何错误，并且把错误绑定到一个名字为 `error` 的局部常量。关于模式匹配的更多信息请参考 模式(../chapter3/07_Patterns.html)。

`catch` 子句不必将 `do` 子句中的代码所抛出的每一个可能的错误都作处理。如果所有 `catch` 子句都未处理错误，错误就会传递到周围的作用域。然而，错误还是必须要被某个周围的作用域处理的——要么是一个外围的 `do-catch` 错误处理语句，要么是一个 throwing 函数的内部。举例来说，下面的代码处理了 `VendingMachineError` 枚举类型的全部枚举值，但是所有其它的错误就必须由它周围的作用域处理：

```
var vendingMachine = VendingMachine()
vendingMachine.coinsDeposited = 8
do {
    try buyFavoriteSnack("Alice", vendingMachine: vendingMachine)
} catch VendingMachineError.InvalidSelection {
    print("Invalid Selection.")
} catch VendingMachineError.OutOfStock {
    print("Out of Stock.")
} catch VendingMachineError.InsufficientFunds(let coinsNeeded) {
    print("Insufficient funds. Please insert an additional \(coinsNeeded) coins.")
}
// 打印 “Insufficient funds. Please insert an additional 2 coins.”
```

上面的例子中， `buyFavoriteSnack(_:vendingMachine:)` 函数在一个 `try` 表达式中调用，因为它能抛出错误。如果错误被抛出，相应的执行会马上转移到 `catch` 子句中，并判断这个错误是否要被继续传递下去。如果没有错误抛出， `do` 子句中余下的语句就会被执行。

将错误转换成可选值

可以使用 `try?` 通过将错误转换成一个可选值来处理错误。如果在评估 `try?` 表达式时一个错误被抛出，那么表达式的值就是 `nil` 。例如,在下面的代码中, `x` 和 `y` 有着相同的数值和等价的含义：

```
func someThrowingFunction() throws -> Int {
}

let x = try? someThrowingFunction()

let y: Int?
do {
    y = try someThrowingFunction()
} catch {
    y = nil
}
```

如果 `someThrowingFunction()` 抛出一个错误， `x` 和 `y` 的值是 `nil` 。否则 `x` 和 `y` 的值就是该函数的返回值。注意，无论 `someThrowingFunction()` 的返回值类型是什么类型， `x` 和 `y` 都是这个类型的可选类型。例子中此函数返回一个整型，所以 `x` 和 `y` 是可选整型。

如果你想对所有的错误都采用同样的方式来处理，用 `try?` 就可以让你写出简洁的错误处理代码。例如，下面的代码用几种方式来获取数据，如果所有方式都失败了则返回 `nil` ：

```
func fetchData() -> Data? {
    if let data = try? fetchDataFromDisk() { return data }
    if let data = try? fetchDataFromServer() { return data }
    return nil
}
```

禁用错误传递

有时你知道某个 throwing 函数实际上在运行时是不会抛出错误的，在这种情况下，你可以在表达式前面写 `try!` 来禁用错误传递，这会把调用包装在一个不会有错误抛出的运行时断言中。如果真的抛出了错误，你会得到一个运行时错误。

例如，下面的代码使用了 `loadImage(_:)` 函数，该函数从给定的路径加载图片资源，如果图片无法载入则抛出一个错误。在这种情况下，因为图片是和应用绑定的，运行时不会有错误抛出，所以适合禁用错误传递：

```
let photo = try! loadImage("./Resources/John Appleseed.jpg")
```

指定清理操作

可以使用 `defer` 语句在即将离开当前代码块时执行一系列语句。该语句让你能执行一些必要的清理工作，不管是以何种方式离开当前代码块的——无论是由于抛出错误而离开，还是由于诸如 `return` 或者 `break` 的语句。例如，你可以用 `defer` 语句来确保文件描述符得以关闭，以及手动分配的内存得以释放。

`defer` 语句将代码的执行延迟到当前的作用域退出之前。该语句由 `defer` 关键字和要被延迟执行的语句组成。延迟执行的语句不能包含任何控制转移语句，例如 `break` 或是 `return` 语句，或是抛出一个错误。延迟执行的操作会按照它们被指定时的顺序的相反顺序执行——也就是说，第一条 `defer` 语句中的代码会在第二条 `defer` 语句中的代码被执行之后才执行，以此类推。

```
func processFile(filename: String) throws {
    if exists(filename) {
        let file = open(filename)
        defer {
            close(file)
        }
        while let line = try file.readline() {
            // 处理文件。
        }
        // close(file) 会在这里被调用，即作用域的最后。
    }
}
```

上面的代码使用一条 `defer` 语句来确保 `open(_:)` 函数有一个相应的对 `close(_:)` 函数的调用。

> 注意
> 即使没有涉及到错误处理，你也可以使用 `defer` 语句。

上一篇：可选链 (/project/swift/chapter2/17_Optional_Chaining.html)
下一篇：类型转换 (/project/swift/chapter2/19_Type_Casting.html)

被顶起来的评论

李杰 (http://t.qq.com/lj5782)


(http://t.qq.com/lj5782)

VendingMachine 类里面的这个字典不能初始化，不知道是为什么。把这个字典单独写到外面也不行。这个Item的结构体不能当做字典的Value吗？？
var inventory = [
"Candy Bar": Item(price: 12, count: 7),
"Chips": Item(price: 10, count: 4),
"Pretzels": Item(price: 7, count: 11)
]
会报exc_bad_access(code=1,address= 这个错误。
吧Item 都改成 " "这样就没问题。

4月18日 回复 顶(1) 转发


(http://weibo.com/123)

Aaron--升 (http://weibo.com/123)
错误抛出，最终以此函数结束，如下：
func buyFavoriteSnack(person: String) throws {
let snackName = favoriteSnacks[person] ?? "Candy Bar"
try vend(itemNamed: snackName)
}

可是，要如何调用此函数呢？本文没有说！我尝试了调用，但是由于此函数可能会抛出错误，所以调用的时候必须要在前面加try，但是非错误抛出函数又无法调用，所以必须是错误抛出函数才能调用错误抛出函数，这么说，所有函数都要加上throws吗？神经病！
举个例子来说，例如我要在viewDidLoad中调用我得这么写，try buyFavoriteSnack("Alice")
可是viewDidLoad不给我调用，viewDidLoad没有throws！难道我要在viewDidLoad函数后面加上throws吗？

真是气死我了，看了几遍都没看懂！

2015年10月9日 回复 顶(1) 转发

27条评论 1条新浪微博 最新 最早 最热


(http://t.qq.com/lj5782)

李杰 (http://t.qq.com/lj5782)
回复 李杰: 经过多次的反复测试，原来是项目命名的问题，项目名字里面包含了数字的话就会出现这个问题。可以吧项目名称修改一下，然后再rename 相关联的所有文件，再跑一次就没问题了。不知道swift 为什么会这样。但是即使文件路径有数字也是没问题的。目前还不清楚为什么会这样，希望有知道原因的大神可以解释一下。

4月20日 回复 顶 转发


(http://t.qq.com/lj5782)

李杰 (http://t.qq.com/lj5782)
回复 李杰: class Temp{
let name = "abc"
}
var tempDic = ["bar": Temp()]
这样子的也不行，也是报那个错误，有人知道为毛吗？

4月18日 回复 顶 转发


(http://t.qq.com/lj5782)

李杰 (http://t.qq.com/lj5782)
VendingMachine 类里面的这个字典不能初始化，不知道是为什么。把这个字典单独写到外面也不行。这个Item的结构体不能当做字典的Value吗？？
var inventory = [
"Candy Bar": Item(price: 12, count: 7),
"Chips": Item(price: 10, count: 4),
"Pretzels": Item(price: 7, count: 11)
]
会报exc_bad_access(code=1,address= 这个错误。
吧Item 都改成 " "这样就没问题。

4月18日 回复 顶(1) 转发

fangry0823
文章翻译的很好。顶

4月7日 回复 顶 转发


(http://weibo.com/3015050631)3月30日

小嘟嘟l (http://weibo.com/3015050631)
好多地方有问题呀，哎 先看看吧，看完了再看视屏去

回复 顶 转发


(http://weibo.com/realank)1月23日

Realank刘 (http://weibo.com/realank)
回复 Aaron--升: 哥们，用do－catch

回复 顶 转发

Andrew
回复 Aaron--升: 你真的有看文章嘛

2015年12月30日 回复 顶 转发


(http://t.qq.com/xclidongbo)2015年12月26日

李东波 (http://t.qq.com/xclidongbo)
回复 李东波: 结贴. 最后加的那个catch.和enum中的default一个屌用...否则会报错.

回复 顶 转发

李东波 (http://t.qq.com/xclidongbo)

在 用 Do-Catch 处理错误 一段落中的demo. 报错如下: Errors thrown from here are not handled because the enclosing catch is not exhaustive.后来在后面加上一句,才编译通过:

```
do {
try buyFavoriteSnack("Alice", vendingMachine: vendingMachine)
} catch VendingMachineError.InvalidSelection {
print("Invalid Selection.")
} catch VendingMachineError.OutOfStock {
print("Out of Stock.")
} catch VendingMachineError.InsufficientFunds(let coinsNeeded) {
print("Insufficient funds. Please insert an additional \(coinsNeeded) coins.")
}

catch {
print("other error")
}
```

定义的三个异常我都捕获了,不知道为什么还给我提示没全面的捕获?

2015年12月26日 回复 顶 转发

TRY美剧 (http://weibo.com/dejavuzhou)

回复 苏服: favoriteSnacks[person] 如果是nil 设置他为 snackName= "CandyBar"
favoriteSnacks[person] 有值 snackName = favoriteSnacks[person]

2015年12月16日 回复 顶 转发

李东波 (http://t.qq.com/xclidongbo)

虽然不知道错误处理,有什么鸟用.不过感谢翻译.

2015年11月30日 回复 顶 转发

吃货终将瘦成闪电 (http://weibo.com/longhaihai)

回复 苏服: Nil Coalescing Operator

The nil coalescing operator (a ?? b) unwraps an optional a if it contains a value, or returns a default value b if a is nil. The expression a is always of an optional type. The expression b must match the type that is stored inside a.

2015年11月27日 回复 顶 转发

苏服

let snackName = favoriteSnacks[person] ?? "Candy Bar"
请教这句什么意思?

2015年11月25日 回复 顶 转发

freeman

回复 Aaron--升: func viewDidLoad()->void {
...
do {
try buyFavoriteSnack("Alice")
}
catch {
}
...
}

2015年11月24日 回复 顶 转发

从今以后 (http://t.qq.com/congjinyih6382)

回复 Aaron--升:

2015年11月20日 回复 顶 转发

TRY美剧 (http://weibo.com/dejavuzhou)

没有使用过异常处理啊!!!!!

2015年11月10日 回复 顶 转发

river (http://t.qq.com/hewansong)

不管怎样 翻译辛苦啦

2015年11月6日 回复 顶 转发

heronlyj (http://weibo.com/lyjonly)

回复 Aaron--升: try! buyFavoriteSnack("Eve")

或者 把 var amountDeposited = 1.00 改成 var amountDeposited = 10.00 之后

```
try! buyFavoriteSnack("随便写asdfad")

或者捕获异常

do {
try vend(itemNamed: "Candy Bar")
```

```
// Enjoy delicious snack
} catch VendingMachineError.InvalidSelection {
print("Invalid Selection")
} catch VendingMachineError.OutOfStock {
print("Out of Stock.")
} catch VendingMachineError.InsufficientFunds(let amountRequired) {
print("Insufficient funds. Please insert an additional $\(amountRequired).")
}
```

2015年10月22日 回复 顶 转发

zlk

回复 Aaron--升: 为你智商捉急啊

2015年10月22日 回复 顶 转发

第三方

回复 锅巴GG不唠嗑: .

2015年10月19日 回复 顶 转发

1 2

(http://

(http://

html)

基础部分 (http://wiki.jikexueyuan.com/project/swift/chapter2/01_The_Basics.html)

基本运算符 (http://wiki.jikexueyuan.com/project/swift/chapter2/02_Basic_Operators.html)

字符串和字符 (http://wiki.jikexueyuan.com/project/swift/chapter2/03_Strings_and_Characters.html)

集合类型 (http://wiki.jikexueyuan.com/project/swift/chapter2/04_Collection_Types.html)

控制流 (http://wiki.jikexueyuan.com/project/swift/chapter2/05_Control_Flow.html)

函数 (http://wiki.jikexueyuan.com/project/swift/chapter2/06_Functions.html)

闭包 (http://wiki.jikexueyuan.com/project/swift/chapter2/07_Closures.html)

Via 由 [极客学院 Wiki

(http://wiki.jikexueyuan.com)]

提供